Review Topics

- JUnit Tests
- Exceptions
- Strings
- Reference Copy, Shallow Copy and Deep Copy
- Array Definitions
- Method definitions
- static and final keywords
- modular arithmetic

JUnit Tests

- Test your code
  - methods/variable definitions/constructors
- Assert Statements
  - assertTrue(boolean condition)
  - assertFalse(boolean condition)
  - assertNotNull(object)
  - assertNull(object)
  - assertEquals(expected, actual, tolerance)

JUnit Tests

- Test your code
  - methods/variable definitions/constructors
- Assert Statements
  - assertTrue(boolean condition)
    - Checks if the boolean condition is true
  - assertFalse(boolean condition)
    - Checks if the boolean condition is false
  - assertNotNull(object)
    - Checks if the object is not Null
  - assertNull(object)
    - Checks if the object is Null
  - assertEquals(expected, actual, tolerance)
    - Check if the expected and actual value are within the specified tolerance.

```
public class InsertElement{
  public static boolean insertElement(int[] intArray,  int pos, int value){
      if(intArray == null){ // if intArray is null
         if(pos!=0) return false;
         intArray = new int[1]{value}; }
      else{
         if(pos>=intArray.length || pos<0) { return false; }
         int[] tempArray = new int[intArray.length+1];
         for(int i = 0, tempIndex=0; i<intArray.length; i++, tempIndex++){
            if(i==pos){ // inserting the new Value at that position
               tempArray[tempIndex] = value;
               tempIndex++; // increment the temp counter after adding the element  }
            tempArray[tempIndex] = intArray[i];
          }
         intArray = tempArray; //reassigning tempArray to intArray
  } return true; }//ending method
}//ending class
```

InsertElement Example:

- inserts the given value into the array at the given position
- updated array is set to the original array reference.
- returns true if successfully inserts the element or else returns false.

Writing tests for a successful insert

```
@Test
public void testInsertSuccess() {
        public int[] testArray = {1, 2, 3, 4, 5, 6};
        boolean isSuccess = InsertElement.insertElement(testArray, 3, 100);
        assertTrue(isSuccess);
        assertEquals(testArray[3], 100);
        assertEquals(testArray[4], 4);
        assertEquals(testArray[2], 3);
        assertTrue(testArray.length==7);
        assertTrue(testArray[testArray.length-1] == 6);
        assertTrue(testArray[0] == 1);
}
```

Writing tests for a Fail insert

```
@Test
public void testInsertFailure() {
        public int[] testArray = new int[6]{1, 2, 3, 4, 5, 6};
        boolean isSuccess = InsertElement.insertElement(testArray, testArray.length+2,
100);
        assertFalse(isSuccess); //assertTrue(isSuccess==false);
        boolean isFailure = InsertElement.insertElement(testArray, -2, 100);
        assertFalse(isFailure==true); // assertTrue(!isFailure);
}
```

Exceptions

- To catch run time errors
- Another way to test the correctness of the code
- Few RunTimeExceptions
  - NumberFormatException (Double.parseDouble())
  - IllegalArgumentException (Calling a function which takes different arguments)
  - NullPointerException (Calling methods on Null Objects)
  - ArrayIndexOutOfBoundsException (Accessing an element outside the index array)

Handling Exceptions

- Throw Exception
  - Code snippet that expects a run time error and throws the exception
- try block around the code that calls the supsect code
  - try{}, catch{}
  - try{}, catch{}, finally{}
- InsertExample: throw Exception if the insert position is less than 0 or greater than the length

```
public class InsertElement{
  public static boolean insertElement(int[] intArray, int pos, int value){
      if(intArray == null){ // if intArray is null
          if(pos!=0) return false;
          intArray = new int[1]{value}; }
      else{
          if(pos>=intArray.length && pos<0) { return false; }
          int[] tempArray = new int[intArray.length+1];
          for(int i = 0, tempIndex=0; i<intArray.length; i++, tempIndex++){
              if(i==pos){ // inserting the new Value at that position
                  tempArray[tempIndex] = value;
                  tempIndex++; // increment the temp counter after adding the element  }
              tempArray[tempIndex] = intArray[i];
            }
          intArray = tempArray; //reassigning tempArray to intArray
  } return true; }//ending method
 }//ending class
```

```
public class InsertElement{
  public static void insertElement(int[] intArray, int pos, int value{
    if(intArray == null){ // if intArray is null
      if(pos!=0) throw new ArrayIndexOutOfBoundsException("pos outside of array
bounds");
      intArray = new int[1]{value}; }
    else{
      if(pos>=intArray.length && pos<0)
        throw new ArrayIndexOutOfBoundsException("pos outside of array bounds");
      int[] tempArray = new int[intArray.length+1];
      for(int i = 0, tempIndex=0; i<intArray.length; i++, tempIndex++){
        if(i==pos){ // inserting the new Value at that position
          tempArray[tempIndex] = value;
          tempIndex++; // increment the temp counter after adding the element  }
        tempArray[tempIndex] = intArray[i];
       }
      intArray = tempArray; //reassigning tempArray to intArray
  }
```

Writing tests for a successful insert //still the same

```
@Test
public void testInsertSuccess() {
        public int[] testArray = new int[6]{1, 2, 3, 4, 5, 6};
        InsertElement.insertElement(testArray, 3, 100);
        assertEquals(testArray[3], 100);
        assertEquals(testArary[2], 3);
        assertTrue(testArray.length==7);
        assertTrue(testArray[testArray.length-1] == 6);
}
```

Writing tests for a successful insert

```
@Test
public void testInsertSuccess() {
        public int[] testArray = new int[6]{1, 2, 3, 4, 5, 6};
        try {
                InsertElement.insertElement(testArray, 3, 100);
                assertEquals(testArray[3], 100);
                assertEquals(testArary[2], 3);
        }
        catch(ArrayIndexOutOfBoundsException e) {
                System.out.println(e.getMessage()); //prints the message indicated in the throw statement
        }
}
```

Writing tests for a Fail insert

```
@Test
public void testInsertFailure() {
        public int[] testArray = new int[6]{1, 2, 3, 4, 5, 6};
        try {
                InsertElement.insertElement(testArray, testArray.length+2, 100);
                InsertElement.insertElement(testArray, -2, 100);
        }
        catch(Exception e) { //Not Recommended
                e.printStackTrace(System.out);// prints the stack trace
        }
}
```

Strings

```
String s= new String(“hello”);
String s1 = “hello”;
assertTrue(s1.charAt(0)==‘h’);
assertTrue(s1.indexOf(‘l’)==2);
assertTrue(s1.length()==5);
String s2 = s1.replace(‘l’, ‘b’);
assertEquals(2, s2.indexOf(‘b’));
String s3 = s1 + “world”;
assertEquals(“helloworld”, s3);
//compareTo, Equals, concat methods
```

Instructor Class

```
public class Instructor {
	public String[ ] courses;
	public Instructor() { //default constructor
		courses = null;
	}
	public Instructor(Instructor another) { //copy constructor
		//assume this works
	}

}
```

Reference vs Shallow vs Deep Copy

Instructor[ ] a = new Instructor[6];

Reference Copy

```
Instructor[ ] a = new Instructor[6];
Instructor[ ] b = a; // Reference Copy
```

Shallow Copy

```
Instructor[ ] a = new Instructor[6];
Instructor[ ] b = new Instructor[a.length];
for(int i = 0; i<a.length; i++) {
    b[i] = a[i];
}
```

Deep Copy

```
Instructor[ ] a = new Instructor[6];
Instructor[ ] b = new Instructor[a.length];
for(int i = 0; i<a.length; i++) {
    b[i] = new Instructor(a[i]);
}
```

Method Definition Templates

- Lets say a dummy method
- No Input and No Output
  - public void dummyMethod1()
- One double Input and No Output
  - public void dummyMethod2(double val)
- One double input and return double
  - public double dummyMethod3(double val)
- Return a boolean with input float array argument
  - public boolean dummyMethod4(float[ ] floatArray)
- Return a new float array with two input arguments: float Array and position(integer)
  - public float[ ] dummyMethod5(float[ ] floatArray, int position)

Calling methods

- No Input and No Output
  - public void dummyMethod1()

    dummyMethod1();
- One double Input and No Output
  - public void dummyMethod2(double val)

    double val= 3.0;

    dummyMethod2(val);
- One double input and return double
  - public double dummyMethod3(double val)

    double val = 3.0;

    double returnVal2 = dummyMethod3(val);

Calling methods

- Return a boolean with input float array argument
  - public boolean dummyMethod4(float[ ] floatArray)

  float[ ] floatArray = new float[3]{0.1, 0.5, 0.6};

  boolean returnBool = dummyMethod4(floatArray);
- Return a new float array with two input arguments: float Array and position(integer)
  - public float[ ] dummyMethod5(float[ ] floatArray, int position)

  float[ ] floatArray = new float[3]{0.1, 0.5, 0.6};

  float[] returnFloatArray = dummyMethod5(floatArray);

```
public class InsertElement{
  public static boolean insertElement(int[] intArray, int value, int pos){
      if(intArray == null){ // if intArray is null
          if(pos!=0) return false;
          intArray = new int[1]{value}; }
      else{
          if(pos>=intArray.length && pos<0) { return false; }
          int[] tempArray = new int[intArray.length+1];
          for(int i = 0, tempIndex=0; i<intArray.length; i++, tempIndex++){
              if(i==pos){ // inserting the new Value at that position
                  tempArray[tempIndex] = value;
                  tempIndex++; // increment the temp counter after adding the element  }
              tempArray[tempIndex] = intArray[i];
            }
          intArray = tempArray; //reassigning tempArray to intArray
  } return true; }//ending method
 }//ending class
```

Note the static identifier

Calling a static method

boolean isSuccess = **InsertElement.**
insertElement(intArray, val, pos);

final variables

- Once initialized, cannot change the value
  - **public final int a = 3; // you cannot change the a value**
- When declared in the class
  - the final variable can be initialized in the constructor
  - but cannot change in any of its methods.

```
public class Test{
        public final int var;
                public Test(int newvar) {
                        var = newvar;
                }
                //cannot change the variable var value anywhere
        }
```

final variables

- final array variables
  - the final variable's array properties can be first initialized in the constructor
  - however its values can be changed.

```
public class Test{
    public final int[] var;
        public Test() {
            var = new int[10] { 0, 1, 1, 2, 3, 4, 5, 7, 8, 9} ;
        }
        //array var cannot be reinitialized. like adding , inserting elements later
        //however the contents of the array can be changed
        //var[5] = 1000; // But not var = new int[11];
        public void changeElement(int pos, int val) {
            var[pos] = val;
        }
}
```